APML - Final Project - Snake

4 במרץ 2019

בן אסף (hivemind), 305432833
יובל יעקבי (yuvalja), 302247077

ייצוג המשחק

תיאור המשחק הוא רביעייה:

- המצב בתור הקודם
  - לוח- מערך דו מימדי המכיל מספרים בטווח $[-1, 9]$
  - מיקום הראש - זוג אינדקסים על הלוח
  - כיוון התנועה - אחד מתוך $\{N, S, E, W\}$
- המהלך בתור הקודם - אחד מתוך $\{L, R, F\}$
- התוצאה של המהלך הזה (reward) - מספר שלם
- המצב בתור הבא (כפי שהגדרנו מצב בתור הקודם)

הדרך שבה בחרנו לקודד את המידע הזה היא כדלקמן:

- מצב
  - גודל הלוח הוא קבוע אבל יכול להיות גדול מאוד (עד כדי 250×250), כלומר יכול להגיד למימד מאוד גדול
    * החיסרון המרכזי במימד כזה גדול הוא שצריך שנצטרך המון דוגמאות כדי ללמוד את כל הקומבינציות של לוחות

    על מנת לעזור לרשת, רצינו למקד אותה במה שחשוב במשחק, וזה הצעדים הבאים ובשביל זה מספיק להסתכל על חלון (קטן יחסית) מסביב לראש, לכן עבור הלמידה תמיד לקחנו חלון מסביב לראש (כלומר הראש במרכז החלון), זה מאפשר לנו ללמוד מספר קטן משמעותית של מצבים ומגדיל את הסיכוי של הרשת להפריד בין עיקר לתפל.

    זהו הפרמטר הראשון של האלגוריתם שלנו - גודל החלון חקרנו בטווח של $[2, 9]$ כדי למצוא גודל אופטימלי.

    כמובן שבמקרה והראש היה בשוליים השלמנו את החלון מהצד השני ע"פ חוקי המשחק.

    את הערכים של הלוח נרמלנו ל$[-1, 1]$

- כיוון התנועה - מהסימטריה של המשחק כיוון התנועה בעצם לא משנה דבר, לכן דאגנו תמיד בתהליך הלמידה "לנרמל" את הכיוון, כלומר סובבנו את הלוח כך שבזמן הלמידה תמיד הראש מסתכל למעלה (N), שוב זה מקטין לנו משמעותית את מספר המצבים האפשריים של המשחק
- מהלך - את המהלך קודדנו רק בעצמאות מספר בין $\{0,1,2\}$
- תוצאת המהלך הקודם - כפי שלמדנו בכיתה, זה בעצם עזר לנו לקבוע האם מהלך הוא טוב או לא וכמה הוא טוב (לתת "פרסים" ו"עונשים" לרשת)

מודל

בחרנו בשיטה של Deep q learning , כלומר רשת נוירונים שעושה הערכה ל q learning.
לאחר קריאה באינטרנט ראינו שתי הרחבות לשיטה הבסיסית:

- **רשת כפולה - double Q learning[1]**

על מנת להבין את האיטאיציה מאחורי הרשת הכפולה, נזכר בQ Learning רגיל:

$$Q(s,a) = r(s,a) + \gamma \max_a Q(S',a)$$

נזכר שבתחילת תהליך הלמידה אנחנו לא ממש יודעים מהם הפעולות הטובות, לכן לבחור את הפעולה שתיתן לנו את הq_value הכי טוב (הוא רועש, עוד לא למדנו אותנו) זה לא בהכרח טוב, מהסיבה הזו אנחנו מפרידים את זה לשתי רשתות:

$$Q(s,a) = r(s,a) + \gamma \underbrace{Q\left(S', \underbrace{argmax_a Q(s',a)}_{q\_net}\right)}_{t\_net}$$

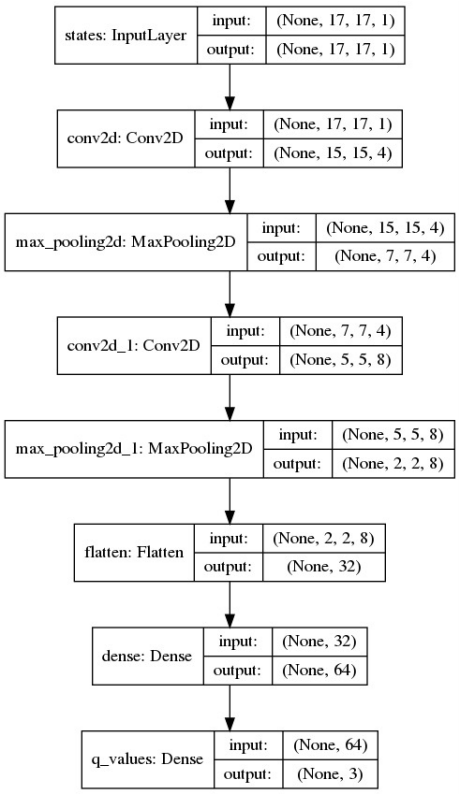
רשת ה q_net משמשת אותנו לQ הפנימי, כלומר מהי הפעולה הכי טובה לעשות עכשיו

רשת הtarget_net על מנת לחשב מהם ערכי הq_values הנכונים

את האימון אנחנו עושים באופן תדיר (כל 5 מהלכים במקרה שלנו) על רשת הq_net וכל m איטרציות (פרמטר של המודל) מעתיקים את המשקולות לt_net.

במקרה שלנו השיטה הזו הורידה מאוד את השונות במהלך תהליך הלמידה

בתכנון הרשת היה לנו tradeoff בין הזמנים שהיינו צריכים לעמוד בהם לבין גודל הרשת, רצינו רשת גדולה יותר כדי שתוכל ללמוד ייצוג טוב יותר של הלוח אבל אז לא עמדנו בזמנים, בסוף זו הארכיטקטורה שהגענו עליה כך שמגיעה לתוצאות טובות וכן עומדת בזמנים:



- **רשתות מתחרות - Dueling DQN** [2]

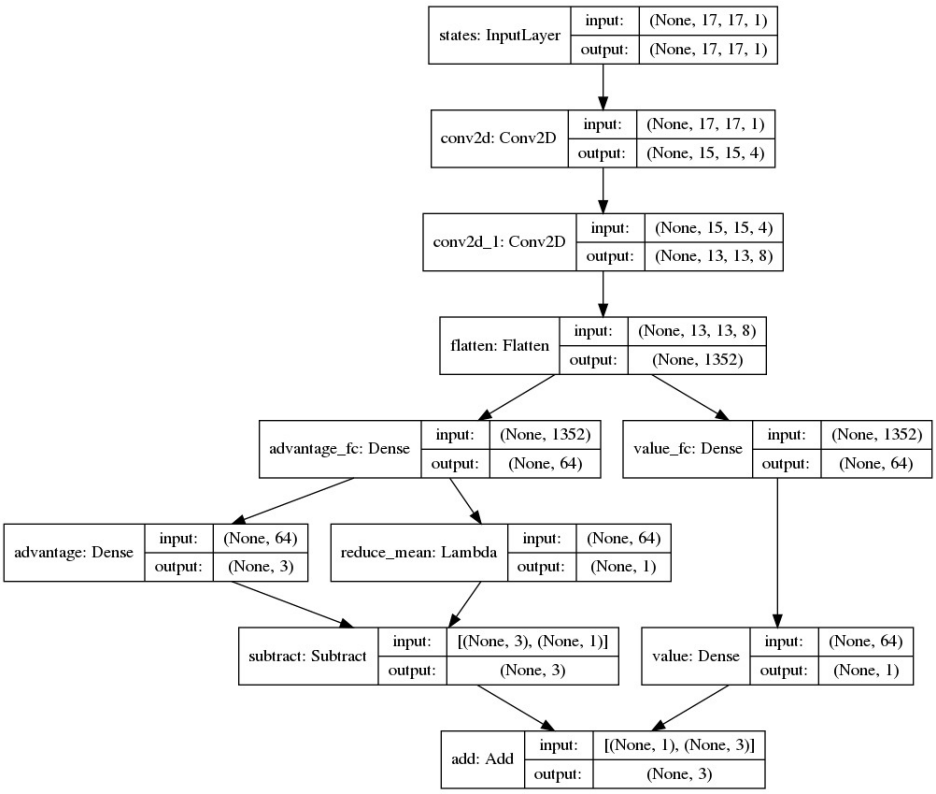
כאמור הq value נועד לתאר: כמה טוב להיות במצב מסוים, וכמה טוב לקחת פעולה מתוך המצב הזה, אז נוכל לפרק אותו בצורה הזו:

$$Q(s,a) = A(s,a) + V(s)$$

בשיטה הזו, אנו בונים שני מסלולים הלומדים בנפרד את שתי הפונקציות A, V ובשכבה האחרונה מאחדים אותם כדי יחד כדי למצוא את ה q value (כפי שניתן לראות בציור לפני ההפרדה למסלולים אנחנו לומדים איזשהו embedding ללוח)

הרעיון מאחורי השיטה זה שיכול להיות שישנם מצבים שפשוט טובים יותר/פחות, בלי קשר לפעולה עצמה (לדוגמה אצלנו, מצב שבו נכנסים להתנגשות וודאית) ואנחנו רוצים ללמוד את המצבים האלה ורק אחרי זה להלביש עליהם את הפעולה הטובה ביותר.

תיאור הרשת:



מבנה הזיכרון

כפי שראינו בכיתה השתמשנו במנגנון הreplay memory, זוהי מחסינת בגודל קבוע (פרמטר של המודל) ששומרת את המצבים האחרונים שראינו במהלך המשחק.

כלומר בכל פעולת act אנחנו שומרים את כל הרביעיה לתוך הזיכרון.

כאשר במקרה של מוות (כלומר המצב החדש נמצא במקום אחר לגמרי מהמצב הקודם) אנחנו שומרים בזיכרון רק את השלישיה: מצב קודם, פעולה ופרס, האינטואיציה מאחורי זה היא ברורה, זה רק יבלבל את המודל ואין סיבה להתייחס למצב החדש כיוון שהוא רנדומלי, אבל מצד שני מאוד חשוב ללמוד על המצבים האלה מכיוון שבהם יש את העונש הכי גדול

בתהליך הלמידה דגמנו באופן אחיד מתוך הזיכרון הזה את גודל הbatch הנדרש (שזה כמובן עוד פרמטר של המודל)

exploration-exploitation trade-of

בחרנו לממש $\epsilon-$greedy policy, כלומר בהסתברות ϵ אנחנו עושים פעולה רנדומית ובהסתברות $1-\epsilon$ אנחנו מכניסים למודל.

רצינו להוריד את כמות הexploration שעושים עם התקדמות הלמידה של המודל (כיוון שהמודל טוב יותר עם הזמן ולכן נרצה לתת לו יותר משקל) ולכן הגדרנו שני פרמטרים למודל ϵ מקסימלי ו ϵ מינמלי, ועשינו אינטרפלוציה לינארית על גבי מספר הסיבובים על מנת לחשב את האפסילון לכל סיבוב.

השיטה הזו מאפשרת לנו בהתחלה לגלות הרבה מצבים ולהבין את המשחק, ולקראת הסוף לקבל את התוצאה הטובה ביותר בלי מהלכים רנדומלים.
פיצלנו את המהלכים הרנדומליים שלנו לשניים:

- בהסתברות $\frac{\epsilon}{2}$ עשינו מהלך רנדומי לגמרי (הסתברות אחידה בין $\{R, F, L\}$)
- בהסתברות $\frac{\epsilon}{2}$ עשינו מהלך חמדן

מה הכוונה במהלך חמדן? הרעיון הוא לחקות את פוליסת הavoid (עם שידרוג קל), בנינו מיפוי בין הסימנים השונים על הלוח ובין ה"פרס" המתאים להם, ואז בכל פעם שבחרנו במהלך חמדן הסתכלנו על שלושת הסימנים האפשריים לקבל במהלך הבא (איזה סימן נקבל מכל אחת מהפעולות) ובחרנו את זו שתניב לנו את הרווח הכי גבוהה ע"פ המיפוי שלנו.

איך בנינו את המיפוי? בכל תור שמרנו את הסימן שהגענו אליו ואת הפרס שקיבלנו כיוון שבחוקי המשחק לא הוגדר שהמיפוי בין סימן ל"פרס" הוא קבוע, ביצענו ממוצע רץ (Moving Average) של כל הפעמיים שראינו את הסימן הזה והפרס המתאים. וזו המפת "סימן - פרס" שלנו.

ניסויים

הערה - קוד תומך לניסויים נמצא בexperiments.py בנוסף לשני הpolicies הרלוונטיים תחת תיקיית policies

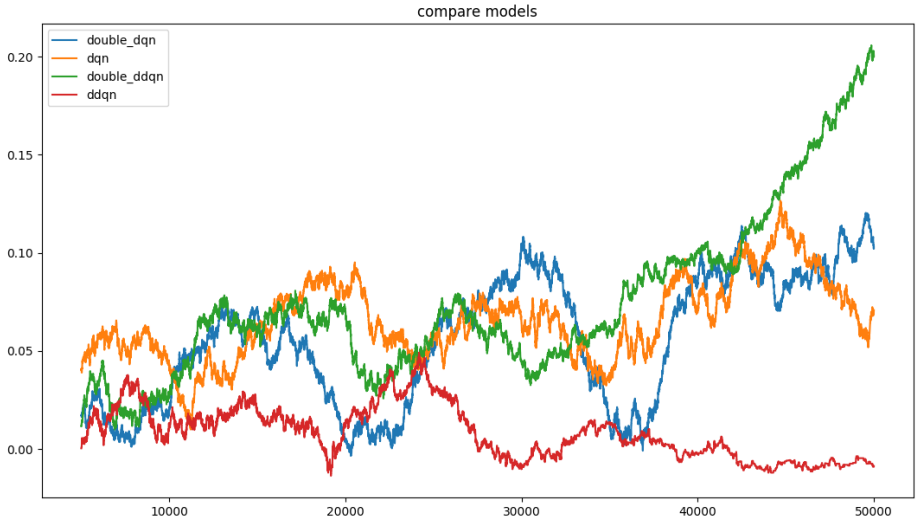
עשינו המון ניסויים לא מתעודים שנועדו לבדוק את הפרמטרים השונים של המודל (החל מגודל החלון, γ, batch_size, learning rate וכולי), את התוצאות שלהם ניתן לראות בפרמטרים שבחרנו

יש לנו גרף של הניסוי הכי חשוב, שבו בחרנו את המודל, כפי שתארינו בחלק הראשון בבנינו שני סוגים של מודלים deep q learning ו dueling deep q learning ולכל אחד מהם אפשר להוסיף את האלמנט של double 2 רשתות שמורידות את השונות.

נתאר את הניסוי:

כשמנו את כל ארבעת המודלים לשחק אחד נגד השני, שמרנו את התוצאות בכל איטרציה (ערכנו את הקוד של Snake.py כך שהוא מחזיר לנו את מערך התוצאות שהוא שומר).

סה"כ קיבלנו $50,000$ תוצאות לכל אחד מארבעת המודלים, עשינו ממוצע רץ בגודל $5,000$ (כמו שיקרה כאשר תעריכו את המודלים שלנו) וזו התוצאה:



מכאן ניתן לראות שהמודל הכי טוב זה DDDqn - double dueling deep q network
וזה המודל שהגשנו כcustom

לינארי

כיוון שהמודל הלינארי קטן ורץ למספר קטן של סיבובים רצינו לבנות מודל הרבה יותר פשוט, מודל ששם הרבה יותר דגש על הפרסים המידיים ולא על הפרסים העתידיים
עשינו את זה בשתי דרכים עיקריות:

- הורדת מימד - השתמשנו ברדיוס של 2 (כלומר גודל הלוח שהרשת לומדת הוא 5×5)
- γ נמוך - הdiscount factor בעצם מכריע כמה דגש לשים על פרסים עתידיים וכמה על הפרסים המיידים, ברגע שהורדנו אותו נמוך כל כך הוא מסתכל בעיקר על הפרסים המידיים.

שמנו לב שהמודל הלינארי נוטה להתנגש בעצמו, כדי לפתור את זה הוספנו מצבים לזיכרון. בכל פעולת act הסתכלנו על שלושת השכנים (בלי קשר לצעד הנבחר) במידה ואחד השכנים היה הנחש שלנו עצמו (נעזרנו בself.id) הוספנו אותו לזיכרון עם $reward = -100$, כדי ללמד את המודל שזו פעולה שמאוד כדאי להימנע ממנה.
באמת ראינו שזה עזר והנחש התנגש פחות בעצמו.
לא הסופנו את הפ'יצר הזה למודל הגדול מכיוון שהוא רואה הרבה מצבים (כי הוא רץ להרבה זמן) והעדפנו לא להוסיף מידע לא אמיתי כשלא צריך

רעיונות נוספים

רעיונות נוספים שניסנו ולא עבדו:

- מיקום הראש - על מנת לא לאבד מידע הוספנו גם את המיקום של הראש לרשת. (לא הוספנו ישירות לשכבה הראשונה, נרחיב על זה בהמשך בתיאור של הרשת).

- ללמוד רק על תוצאות משמעותיות - הרעיון דומה success learning שראינו בApproxiPong: נכניס לזיכרון (כלומר למצבים שמהם הרשת לומדת) רק מצבים בהם התוצאה שונה מ0, הסיבה לכך היא שרוב המצבים בעצם נותנים לנו reward של 0 ומעט מצבים מעניינים שמהם היינו רוצים ללמוד להימנע/לעשות אותם. כאשר עשינו את זה הרשת לא הצליחה ללמוד בכלל הסיבה שחשבנו עליה זה שהיו מעט מידי מצבים בזיכרון והוא לא הצליח להכליל כמו שצריך

- לשנות את הreward כאשר אין פרס - ראינו שלפעמים כאשר ϵ כבר יחסית קטן הנחש מחליט במשך כמה תורות רצוף לעשות F (זו הייתה בחירה ולא דיפולט), אז על פעולות ללא reward שמרנו בזיכרון -10^{-5} במקום 0, כלומר מספר מאוד מאוד קטן על מנת לעודד את הרשת לא לבחור בפעולה הזו אלא לנסות לעשות דברים אחרים. בפועל קיבלנו תוצאות משמעותית פחות טובות

רשימת מקורות

[1] Van Hasselt, Hado, Guez, Arthur, and Silver, David. Deep reinforcement learning with double q-learning. arXiv preprint arXiv:1509.06461, 2015.

[2] Wang, Z., de Freitas, N., and Lanctot, M. Dueling Network Architectures for Deep Reinforcement Learning. ArXiv e-prints, November 2015.